Event Types explained

There are 14 event types:

- GameStartedEvent
- GameProgressionChangedEvent
- ConversationFinishedEvent
- RoomUnlockedEvent
- ComponentTestsActivatedEvent
- MutatedComponentTestsFailedEvent
- ComponentDestroyedEvent
- DebugStartEvent
- ComponentTestsExtendedEvent
- ComponentFixedEvent
- cut-modified
- test-modified
- test-executed
- test-execution-failed

0. General event info

```
{
    "id": 9971, // Each event has an unique, sequential ID.
    "user": "ovchinin663", // the user associated with the event
    "timestamp": 1716038307036, // the timestamp of the point in time the event is written to the database
    "eventType": "GameStartedEvent", // the event type. One of the types listed above
    "data": {
        // the serialized version of the event
    }
}
```

1. GameStartedEvent

The GameStartedEvent is sent from the client to the server, indicating that the game has fully loaded and the client is ready to recieve events. The server will then respond with a GameProgressionChangedEvent, that indicates the current game state.

```
{
    "id": 9971,
    "user": "ovchinin663",
    "timestamp": 1716038307036,
    "eventType": "GameStartedEvent",
    "data": {
        "type": ".GameStartedEvent", // Java class name of the event
        "timestamp": 1716038307033 // time when the event was created & sent
    }
}
```

2. GameProgressionChangedEvent

```
{
    "id": 9966,
    "user": "ross637",
    "timestamp": 1715956536505,
    "eventType": "GameProgressionChangedEvent",
    "data": {
        "type": ".GameProgressionChangedEvent",
        "timestamp": 1715956536503,
        "progression": {
            "id": 3, // the game progression order (same as room id as no second stage was used in the sessions)
            "room": 3, // the room id, starts at 1
            "componentName": "GreenHouse", /* the target component for tests/alarm/etc.
```

```
                                                 * Is always the one component in the room as currently
                                                 * every room has exactly one component
                                                 */
            "stage": 1, // multiple stages where implemented, but not included. This will always be 1.
            "status": "TEST" // the new state of the component/room
        }
    }
},
```

The status can be one of the following:

Status	Description
DOOR	The player has to fix/unlock the door to enter this room
TALK	The player has to talk to the robot to get to the next status (=TEST)
TEST	The player can open the component and write tests for it
TESTS_ACTIVE	The player has activated the tests and can now run around and interact with objects in the current room
DESTROYED	The tests did not detect the mutation and the component was destroyed
MUTATED	The tests did detect the mutation and the alarm is now displayed
DEBUGGING	The player is debugging the mutated CUT

The game progression at room/component level is defined in the `resources/game/game-progression.csv` and loaded automatically in the DB table `game_progression_entity`.
For the class sessions the following game progression order was used:

Order Index	Room ID	Component Name	Stage	Delay Seconds
1	1	CryoSleep	1	15
2	2	Engine	1	40
3	3	GreenHouse	1	20
4	4	Kitchen	1	30
5	5	ReactorLog	1	20
6	6	DefenseSystem	1	20
7	7	RnaAnalyzer	1	20

3. ConversationFinishedEvent

indicates that the user finished a dialogue. Is used to switch from the TALK to the TEST status.
(Sent by the client)

4. RoomUnlockedEvent

Signals that a room was unlocked/ a puzzle was solved. Includes the `"roomId"` parameter that indicates the room that was unlocked.
(Sent by the client)

5. ComponentTestsActivatedEvent

Signals that the user activated the tests they wrote for the component with the name `"componentName"`.
(Sent by the client)

6. MutatedComponentTestsFailedEvent

Is sent from the server to the client once a component was mutated and the tests written by the user failed and detected the mutation.

```
{
    "id": 9771,
    "user": "ovchinin663",
    "timestamp": 1715865724974,
    "eventType": "MutatedComponentTestsFailedEvent",
    "data": {
        "type": ".MutatedComponentTestsFailedEvent",
        "timestamp": 1715865724950,
        "componentName": "GreenHouse", // the mutated component
        "executionResult": {
            // the executionResult that is generated when running the users tests against the mutated CUT
            // see -> #ExecutionResult explained
            // "testStatus" will always be FAILED for this event
        },
        "cutSource": {
            "cutComponentName": "GreenHouse", // the name of the component this CUT belongs to
            "className": "GreenHouse", // the name of the class
            "sourceCode": "public class GreenHouse {...}\r\n" // the complete source code of the CUT as string
            "editable": [ // the editable lines as a Range object. Is omitted in the prettified version.
                ...
            ]
        },
        "testSource": {
            "cutComponentName": "GreenHouse",
            "className": "GreenHouseTest", // the name of the test class. Is always the CUT name + "Test"
            "sourceCode": "import java.util.*;\n ...", // the complete source code of the users Test as string
            "editable": [ // the editable lines as a Range object. Is omitted in the prettified version.
                {
                    "startLine": 17,
                    "startColumn": 1,
                    "endLine": 72,
                    "endColumn": 1
                }
            ]
        }
    }
},
```

ExecutionResult explained

```
"executionResult": {
    "testClassName": "EngineTest", // the name of the test class that is executed
    "testStatus": "FAILED", // can be of the following: PASSED, FAILED, IGNORED (never occurred)
    "elapsedTime": 1, /* execution time of the test in ms (would be relevant for infinite loops,
                       * but no one even tried to create one. So the only times where the execution
                       * time differs by a large amount is for an InvalidTestClassError that
                       * appeared 7 times becuase players didn't make their test classes public.)
                       */
    "testDetails": { // details of each test method execution:
        "get02test2": {
            "className": "EngineTest", // the class this method is in
            "methodName": "get02test2", // the test class method name
            "testSuiteName": "EngineTest", // this should always be equal to the class name
            "testStatus": "PASSED", // whether the method FAILED or PASSED (IGNORED never occured)
            "actualTestResult": null, // actual and expected values in the case of an
                                      // AssertionError (see next method) or null else
            "expectedTestResult": null,
            "trace": null, // error information, stack trace in case an error occurred (can be a
                           // failed assertion or any other exception)
            "accessDenied": null, /* if the class loader blocked the access to a not-whitelisted or
                                   * blacklisted class, e.g. "Access to the class \"java.lang.System\" was denied.".
                                   * Then the trace will contain a SecurityException.
                                   */
            "startTime": 1715865075041, // timestamp when the execution started
            "elapsedTime": 0
        },
        "hiddenTest4": { // example for a failed test with differing actual/expected values
            "className": "EngineTest",
            "methodName": "hiddenTest4",
            "testSuiteName": "EngineTest",
            "testStatus": "FAILED",
            "actualTestResult": "0.0",
            "expectedTestResult": "1.77272727",
            "trace": "java.lang.AssertionError: expected:<1.77272727> but was:<0.0>\n\tat org.junit.Assert.fail(Assert.java:89)\n\tat org.
            "accessDenied": null,
            "startTime": 1715865075041,
            "elapsedTime": 1
        }
    },
    "coveredLines": {
        "Engine#hobaugh650": 6, // the test suite covered 6 lines
        "CryoSleep#hobaugh650": 0
    },
    "totalLines": {
        "Engine#hobaugh650": 10, // the class tested by this suite has 10 lines
        "CryoSleep#hobaugh650": 0
    },
    "coverage": { // detailed coverage information: maps the line number to the amount of visits
        "Engine#hobaugh650": {
            "14": 2,
            "17": 2,
            "18": 2,
            "19": 2,
            "27": 2,
            "35": 2
        },
        "CryoSleep#hobaugh650": {}
    },
    "variables": { // (example taken from a different event => class/user not consistent)
        "Kitchen#strekalov246": { // maps the line number of a variable change to the list of variables
            "13": {
                // method name (= scope of the variable) "/" variable name : value
                "removeIngredient/current": "0",
                "cookRecipe/availableAmount": "10",
                "addIngredient/current": "0"
            },
            "31": {
                "cookRecipe/success": "1"
            },
            "32": {
                "cookRecipe/entry": "Test=10"
            },
            "34": {
                "cookRecipe/ingredient": "Test2"
            },
            "35": {
                "cookRecipe/requiredAmount": "10"
            },
            "38": {
                "cookRecipe/success": "0"
            },
            "43": {
                "cookRecipe/entry": "Test2=10"
            }
        },
    },
    "logs": {
        "Engine#hobaugh650": [ // lists the logs that the player "printed" out
            {
                "orderIndex": 0, // sequential ID to keep track of the order in which the logs were printed
                "message": "Engine started", // the text content that was logged
                "methodName": "<init>", // the method this log statement belongs to. In this case the constructor.
                "lineNumber": 18, // the line number of the log statement. (used to show logs in the monaco editor)
                "testMethodName": null /* the test method name that led to this log output.
                                        * If null like in this case, the log was called in static context outside of
                                        * a test method (= initializing the engine object at the start of the class
                                        * as a member variable.)
                                        */
            },
            {
                "orderIndex": 1,
                "message": "Engine started",
                "methodName": "<init>",
                "lineNumber": 18,
                "testMethodName": "hiddenTest4"
            }
        ],
        "CryoSleep#hobaugh650": []
    },
    "hiddenTestsPassed": false /* whether the hidden tests PASSED/FAILED for this CUT.
                                * Is also false if the hidden tests weren't executed, because the user tests failed
                                * or because the CUT is not modified and the hidden tests would always pass.
                                */
}
```

7. ComponentDestroyedEvent

This event is sent from the server to the client when the players tests passed on the mutated CUT and therefore the component was destroyed. Additionally to the `"executionResult"` (see previous section) it also contains the source code of the mutated CUT (`"cutSource"`) (so that it can be shown in the editor directly) and the source code of the hidden tests (`"autoGeneratedTestSource"`) (because originally they were used as tests directly. They are no longer relevant for the game now.)

8. DebugStartEvent

The client sends this to the server once the player starts with debugging (after opening the component in a Destroyed/Mutated state and pressing the "Start Debugging" button).
It only includes the `"componentName"` of the component that is now going to be debugged.

9. ComponentTestsExtendedEvent

```
{
    "id": 8544,
    "user": "ivanishin666",
    "timestamp": 1715864020944,
    "eventType": "ComponentTestsExtendedEvent",
    "data": {
        "type": ".ComponentTestsExtendedEvent",
        "timestamp": 1715864020943,
        "componentName": "Engine",
        "addedTestMethodName": "hiddenTest4"
    }
},
```

The `"componentName"`s test class was extended by the method named `"addedTestMethodName"`. This is sent from the server to the client to show the player a modal notification if appropriate and trigger a refresh of the cached test class source code.
Originally it was only sent during debugging when a player changed the component in a way that fixed their own tests, but not the hidden tests. It's now also used alongside the ComponentDestroyedEvent to indicate which method was added to the players test code (as not the whole test suite is replaced with the hidden tests).

10. ComponentFixedEvent

The server sends this to client if the last execution fixed the component (= all player tests & hidden tests passed). Includes only the `"componentName"`.

11. cut-modified

Logs the modification of a CUT by a user. The patch string (example below) describes the difference between the user modified source and the original CUT (before/without the mutation). An empty string means no modification = the player restored the original CUT perfectly.

```
"patch": "@@ -1032,19 +1032,31 @@\n  return \n+Math.floor(\n ch4\n+)\n  * o2toC\n",
```

This is stored when the client sends a PUT request to `api/components/{componentName}/cut/src` and the new modification differs from the one saved before.
Using the patch notation the changes are easier to see. To recieve the complete source code, the diff_match_patch library (see https://github.com/google/diff-match-patch) can be used.

12. test-modified

Logs the modification of a test. This happens either by a PUT request to `api/components/{componentName}/test/src` or by extending the test with a hidden test on execution.
The complete new `"sourceCode"` of the changed test is provided as data.

13. test-executed

Logs the execution result of a test. This happens by a POST request to `api/components/{componentName}/test/execute` (while in TEST or DEBUGGING status).

The data provided is of the type TestExecutionResult (=> See #ExecutionResult explained).

14. test-execution-failed

Logs the failure of a test execution (by POST `api/components/{componentName}/test/execute` during TEST/DEBUGGING), indicating a compilation error. The provided data contains the error message, e.g.:

```
{
    "id": 9958,
    "user": "ross637",
    "timestamp": 1715956303704,
    "eventType": "test-execution-failed",
    "data": "Compilation failed.\n\n/GreenHouseTest.java:49: error: cannot find symbol\n        expected = gh.plants;\n        ^\n  symbol
},
```